



## DOMAIN WINTER WINNING CAMP ASSIGNMENT

**Student Name: Ishu**

**UID: 22BCS15695**

**Branch: BE-CSE**

**Section/Group: 22BCS\_FL\_IOT-603/B**

**Semester: 5<sup>th</sup>**

### Day 8 : Dynamic Programming

**Very Easy:**

#### 1. N-th Tribonacci Number

The Tribonacci sequence  $T_n$  is defined as follows:

$T_0 = 0, T_1 = 1, T_2 = 1$ , and  $T_{n+3} = T_n + T_{n+1} + T_{n+2}$  for  $n \geq 0$ .

Given  $n$ , return the value of  $T_n$ .

**Example 1:**

**Input:  $n = 4$**

**Output: 4**

**Explanation:**

$T_3 = 0 + 1 + 1 = 2$

$T_4 = 1 + 1 + 2 = 4$

**Example 2: Input:  $n = 25$**

**Output: 1389537**

**Constraints:  $0 \leq n \leq 37$**

The answer is guaranteed to fit within a 32-bit integer, ie.  $\text{answer} \leq 2^{31} - 1$ .

**URL-**<https://leetcode.com/problems/n-th-tribonacci-number/description/?envType=problem-list-v2&envId=dynamic-programmin>

**CODE:**

```
#include <iostream>
using namespace std;

int tribonacci(int n) {
    if (n == 0) return 0;
    if (n == 1 || n == 2) return 1;

    int t0 = 0, t1 = 1, t2 = 1;
```

```
int t3 = 0;

for (int i = 3; i <= n; ++i) {
    t3 = t0 + t1 + t2;
    t0 = t1;
    t1 = t2;
    t2 = t3;
}

return t2;
}

int main() {
    cout << tribonacci(4) << endl; // Output: 4
    cout << tribonacci(25) << endl; // Output: 1389537
    return 0;
}
```

**OUTPUT:**

```
4
1389537
```

**Easy:**

## 2. Climbing Stairs

You are climbing a staircase. It takes  $n$  steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

**Example 1: Input:  $n = 2$**

**Output: 2**

**Explanation:** There are two ways to climb to the top.

1. 1 step + 1 step
2. 2 steps

**Example 2: Input:  $n = 3$**

**Output: 3**

**Explanation:** There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

**Constraints:**  $1 \leq n \leq 45$



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

[v2&envId=dynamic-programming](#)

## CODE:

```
#include <iostream>
using namespace std;

int climbStairs(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;

    int prev1 = 1, prev2 = 2;
    for (int i = 3; i <= n; ++i) {
        int current = prev1 + prev2;
        prev1 = prev2;
        prev2 = current;
    }
    return prev2;
}

int main() {
    cout << climbStairs(2) << endl; // Output: 2
    cout << climbStairs(3) << endl; // Output: 3
    cout << climbStairs(10) << endl; // Example for larger input
    return 0;
}
```

## OUTPUT:

```
2
3
89
```

## Medium:

### 3. Longest Palindromic Substring

Given a string *s*, return the longest palindromic substring in *s*.

**Example 1: Input: *s* = "babad"**

**Output: "bab"**

**Explanation: "aba" is also a valid answer.**

**Example 2: Input: *s* = "cbbd"**

**Output: "bb"**

**Constraints:  $1 \leq s.length \leq 1000$**



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

s consist of only digits and English letters.

URL- <https://leetcode.com/problems/longest-palindromic-substring/description/?envType=problem-list-v2&envId=dynamic-programming>

**CODE:**

```
#include <iostream>
#include <string>
using namespace std;

string longestPalindrome(string s) {
    int n = s.size();
    if (n == 0) return "";

    int start = 0, maxLength = 1;

    auto expandAroundCenter = [&](int left, int right) {
        while (left >= 0 && right < n && s[left] == s[right]) {
            int currentLength = right - left + 1;
            if (currentLength > maxLength) {
                start = left;
                maxLength = currentLength;
            }
            --left;
            ++right;
        }
    };

    for (int i = 0; i < n; ++i) {
        expandAroundCenter(i, i);    // Odd-length palindrome
        expandAroundCenter(i, i + 1); // Even-length palindrome
    }

    return s.substr(start, maxLength);
}

int main() {
    cout << longestPalindrome("babad") << endl; // Output: "bab" or "aba"
    cout << longestPalindrome("cbdd") << endl;  // Output: "bb"
    return 0;
}
```

**OUTPUT:**

```
bab
bb
```

**Hard:**

## 4. Maximal Rectangle

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

**Example-**

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

**Input:** matrix=

[[ "1", "0", "1", "0", "0"], [ "1", "0", "1", "1", "1"], [ "1", "1", "1", "1", "1"], [ "1", "0", "0", "1", "0"]]

**Output:** 6

**Explanation:** The maximal rectangle is shown in the above picture.

**Example 2:** Input: matrix = [[ "0" ]]

**Output:** 0

**Example 3:** Input: matrix = [[ "1" ]]

**Output:** 1

**Constraints:**

rows == matrix.length

cols == matrix[i].length

1 <= row, cols <= 200

matrix[i][j] is '0' or '1'.

**URL-**<https://leetcode.com/problems/maximal-rectangle/description/?envType=problem-list-v2&envId=dynamic-programming>

**CODE:**

```
#include <iostream>
```

```
#include <vector>
```

```
#include <stack>
```

```
using namespace std;
```

```
int largestRectangleArea(vector<int>& heights) {
```

```
    stack<int> s;
```

```
    heights.push_back(0); // Append a zero to handle remaining elements
```

```
    int maxArea = 0;
```

```
for (int i = 0; i < heights.size(); ++i) {
    while (!s.empty() && heights[i] < heights[s.top()]) {
        int h = heights[s.top()];
        s.pop();
        int width = s.empty() ? i : (i - s.top() - 1);
        maxArea = max(maxArea, h * width);
    }
    s.push(i);
}
return maxArea;
}

int maximalRectangle(vector<vector<char>>& matrix) {
    if (matrix.empty()) return 0;

    int rows = matrix.size(), cols = matrix[0].size();
    vector<int> heights(cols, 0);
    int maxArea = 0;

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            heights[j] = (matrix[i][j] == '1') ? heights[j] + 1 : 0;
        }
        maxArea = max(maxArea, largestRectangleArea(heights));
    }
    return maxArea;
}

int main() {
    vector<vector<char>> matrix = {
        {'1', '0', '1', '0', '0'},
        {'1', '0', '1', '1', '1'},
        {'1', '1', '1', '1', '1'},
        {'1', '0', '0', '1', '0'}
    };
    cout << maximalRectangle(matrix) << endl; // Output: 6

    vector<vector<char>> matrix2 = {{'0'}};
    cout << maximalRectangle(matrix2) << endl; // Output: 0

    vector<vector<char>> matrix3 = {{'1'}};
    cout << maximalRectangle(matrix3) << endl; // Output: 1

    return 0;
}
```

**OUTPUT:**

6  
0  
1

**Very Hard:**

## 5. Cherry Pickup

You are given an  $n \times n$  grid representing a field of cherries, each cell is one of three possible integers. 0 means the cell is empty, so you can pass through,

1 means the cell contains a cherry that you can pick up and pass through, or

-1 means the cell contains a thorn that blocks your way.

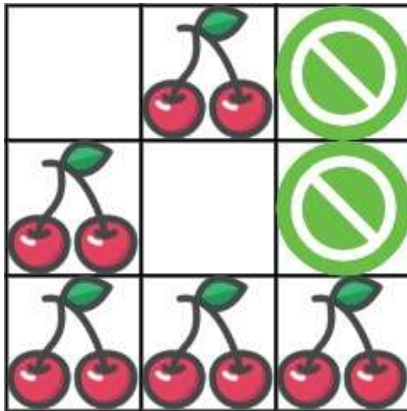
Return the maximum number of cherries you can collect by following the rules below:

Starting at the position (0, 0) and reaching (n - 1, n - 1) by moving right or down through valid path cells (cells with value 0 or 1).

After reaching (n - 1, n - 1), returning to (0, 0) by moving left or up through valid path cells.

When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell 0.

If there is no valid path between (0, 0) and (n - 1, n - 1), then no cherries can be collected.



**Input:** grid = [[0,1,-1],[1,0,-1],[1,1,1]]

**Output:** 5

**Explanation:** The player started at (0, 0) and went down, down, right right to reach (2, 2).

4 cherries were picked up during this single trip, and the matrix becomes [[0,1,-1],[0,0,-1],[0,0,0]].

Then, the player went left, up, up, left to return home, picking up one more cherry.

The total number of cherries picked up is 5, and this is the maximum possible.

**Example 2:** Input: grid = [[1,1,-1],[1,-1,1],[-1,1,1]]

**Output:** 0

**Constraints:**



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
n == grid.length
n == grid[i].length
1 <= n <= 50
grid[i][j] is -1, 0, or 1.
grid[0][0] != -1
grid[n - 1][n - 1] != -1
```

URL-<https://leetcode.com/problems/cherry-pickup/description/?envType=problem-list-v2&envId=dynamic-programming>

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits> // Include this for INT_MIN

using namespace std;

int cherryPickup(vector<vector<int>>> & grid) {
    int n = grid.size();
    vector<vector<vector<int>>> dp(n, vector<vector<int>>(n, vector<int>(n, INT_MIN)));
    dp[0][0][0] = grid[0][0];

    for (int t = 1; t <= 2 * (n - 1); ++t) {
        for (int x1 = min(n - 1, t); x1 >= max(0, t - (n - 1)); --x1) {
            for (int x2 = min(n - 1, t); x2 >= max(0, t - (n - 1)); --x2) {
                int y1 = t - x1, y2 = t - x2;
                if (y1 >= n || y2 >= n || grid[x1][y1] == -1 || grid[x2][y2] == -1) continue;

                int cherries = grid[x1][y1];
                if (x1 != x2) cherries += grid[x2][y2];

                int bestPrev = INT_MIN;
                for (int dx1 : {0, -1}) {
                    for (int dx2 : {0, -1}) {
                        int px1 = x1 + dx1, px2 = x2 + dx2;
                        if (px1 >= 0 && px2 >= 0 && px1 < n && px2 < n) {
                            bestPrev = max(bestPrev, dp[px1][t - px1][px2]);
                        }
                    }
                }

                if (bestPrev != INT_MIN) {
                    dp[x1][y1][x2] = bestPrev + cherries;
                }
            }
        }
    }
}
```





# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return max(0, dp[n - 1][n - 1][n - 1]);
    }

    int main() {
        vector<vector<int>>> grid1 = {
            {0, 1, -1},
            {1, 0, -1},
            {1, 1, 1}
        };
        cout << cherryPickup(grid1) << endl; // Output: 5

        vector<vector<int>>> grid2 = {
            {1, 1, -1},
            {1, -1, 1},
            {-1, 1, 1}
        };
        cout << cherryPickup(grid2) << endl; // Output: 0

        return 0;
    }
```

**OUTPUT:**

5

0